Introduction:

In today's dynamic business landscape, the effective management of data and transactions is crucial for ensuring the smooth operation of financial institutions. This report explores the application of Object-Oriented Programming (OOP) concepts in business solutions through the development of a Java application, showcasing how OOP principles such as encapsulation, inheritance, polymorphism, and abstraction are leveraged to create a modular and efficient application tailored to the banking industry's requirements. Through concise code snippets and explanations, this report highlights the practical advantages of OOP in enhancing the efficiency and maintainability of business applications, underscoring Java's pivotal role in driving innovation in software development for business solutions.

Design:

The Java application developed for the Bank Management System provides a robust solution for managing accounts and transactions within financial institutions. This system leverages Object-Oriented Programming (OOP) principles to ensure flexibility, scalability, and maintainability. The classes involved in this implementation:

Bank: Manages accounts and transactions, providing methods for account operations.

Account (Abstract): Parent class for specific account types, containing common attributes and abstract methods for deposit and withdrawal.

CheckingAccount (Subclass of Account): Represents checking accounts with transaction-specific methods and fee handling.

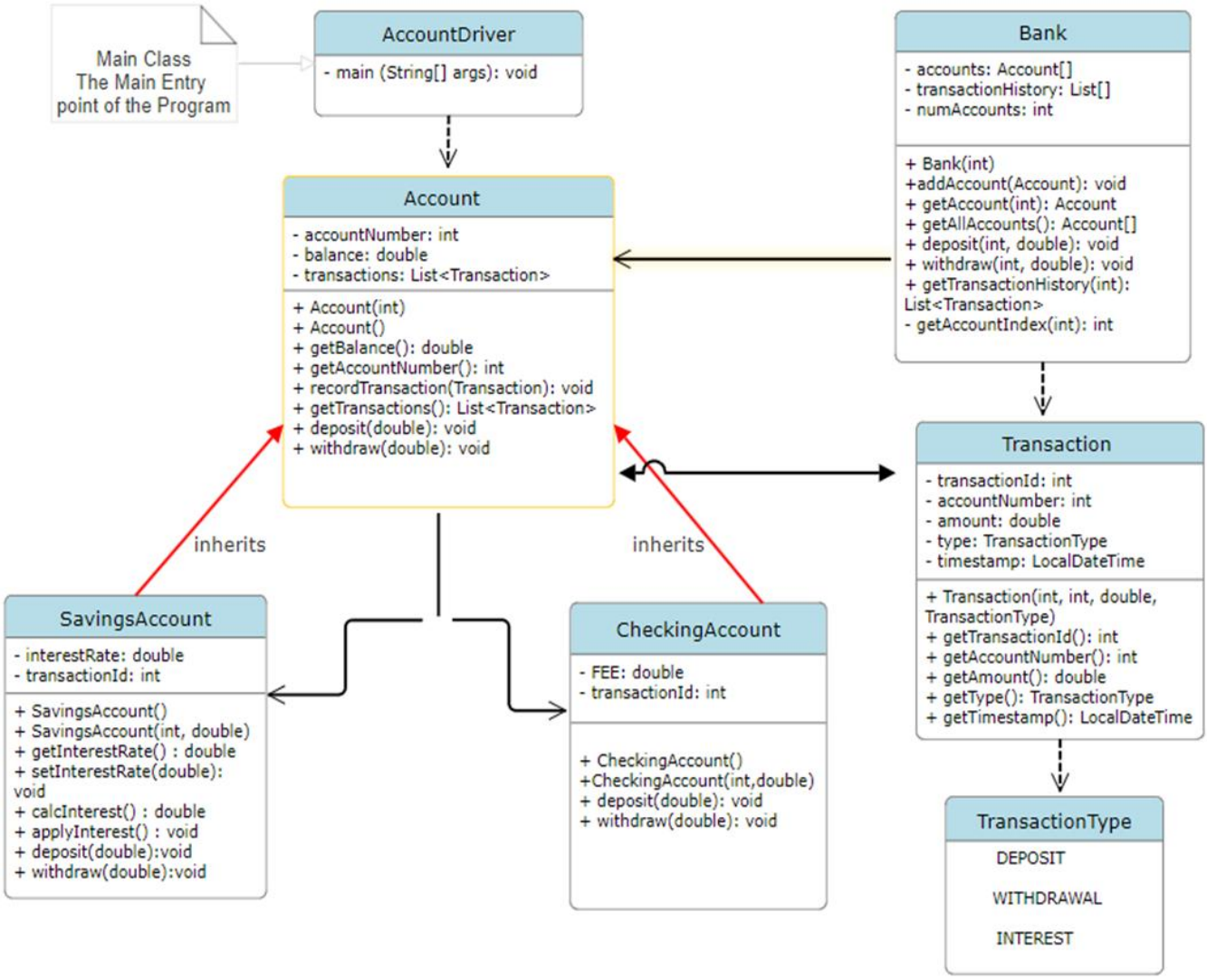
SavingsAccount (Subclass of Account): Represents savings accounts with interest rate calculation and account-specific methods.

Transaction: Manages individual transactions with attributes such as ID, type, amount, and timestamp.

TransactionType Enum: Defines transaction types for categorization.

AccountDriver: Main class for user interaction, implementing menus and input handling for account operations.

Diagram:


Main Class
The Main Entry point of the Program
AccountDriver
- main (String[] args): void
Bank
- accounts: Account[]
- transactionHistory: List[]
- numAccounts: int
+ Bank(int)
+addAccount(Account): void
+ getAccount(int): Account
+ getAllAccounts(): Account[]
+ deposit(int, double): void
+ withdraw(int, double): void
+ getTransactionHistory(int): List<Transaction>
- getAccountIndex(int): int
Account
- accountNumber: int
- balance: double
- transactions: List<Transaction>
+ Account(int)
+ Account()
+ getBalance(): double
+ getAccountNumber(): int
+ recordTransaction(Transaction): void
+ getTransactions(): List<Transaction>
+ deposit(double): void
+ withdraw(double): void
Transaction
- transactionId: int
- accountNumber: int
- amount: double
- type: TransactionType
- timestamp: LocalDateTime
+ Transaction(int, int, double, TransactionType)
+ getTransactionId(): int
+ getAccountNumber(): int
+ getAmount(): double
+ getType(): TransactionType
+ getTimestamp(): LocalDateTime
inherits
inherits
SavingsAccount
- interestRate: double
- transactionId: int
+ SavingsAccount()
+ SavingsAccount(int, double)
+ getInterestRate() : double
+ setInterestRate(double): void
+ calcInterest() : double
+ applyInterest() : void
+ deposit(double):void
+ withdraw(double):void
CheckingAccount
- FEE: double
- transactionId: int
+ CheckingAccount()
+CheckingAccount(int,double)
+ deposit(double): void
+ withdraw(double): void
TransactionType
DEPOSIT
WITHDRAWAL
INTEREST

Code

In this section, we provide the complete implementation of our banking management system. The system is designed to handle various banking operations including account management and transaction handling. The code is organized into multiple classes, each serving a specific purpose in the overall system. Below is an outline of the classes and their functionalities:

Main Class (AccountDriver)

```
import java.util.Scanner;
import java.util.List;
public class AccountDriver {

   // Entry point of program
   public static void main (String [] args) {

      Scanner keyboard = new Scanner (System.in);

      // Create Bank object to manage accounts
      Bank bank = new Bank (100000);

      int choice;

      do {
            choice = menu(keyboard);
         System.out.println();
```

```
    switch(choice) {
      case 1:
        bank.addAccount(createAccount(keyboard));
        break;
      case 2:
        doDeposit(bank, keyboard);
        break;
      case 3:
        doWithdraw(bank, keyboard);
        break;
      case 4:
        applyInterest(bank, keyboard);
        break;
      case 5:
        printTransactionHistory(bank, keyboard);
        break;
      case 6:
        System.out.println("Thank You for using CITY BANK !!!");
        break;
      default:
        System.out.println("Invalid choice. Please try again.");
        break;
    }
    System.out.println();
  } while (choice! = 6);
```

```
    keyboard.close(); // Close the scanner
}
public static int accountMenu(Scanner keyboard) {
    System.out.println("Select Account Type");
    System.out.println("1. Checking Account");
    System.out.println("2. Savings Account");

    int choice;
    do {
        System.out.print("Enter choice: ");
        choice = keyboard.nextInt();
    } while (choice < 1 || choice > 2);

    return choice;
}

/**
 * Function to perform Deposit on a selected account
 */

public static void doDeposit(Bank bank, Scanner keyboard) {
    // Get the account number
    System.out.print("\nPlease enter account number: ");
    int accountNumber = keyboard.nextInt();

    Account account = bank.getAccount(accountNumber);
    if (account! = null) {
```

```
        // Amount
        System.out.print("Please enter Deposit Amount: ");
        double amount = keyboard.nextDouble();
        account.deposit(amount);
        System.out.printf("Deposit of $%.2f successfully made to account %d%n", amount,
accountNumber);
      } else {
        System.out.println("No account exist with AccountNumber: " + accountNumber);
      }
    }

    public static void doWithdraw(Bank bank, Scanner keyboard) {
      // Get the account number
      System.out.print("\nPlease enter account number: ");
      int accountNumber = keyboard.nextInt();

      Account account = bank.getAccount(accountNumber);
      if (account! = null) {
        // Amount
        System.out.print("Please enter Withdraw Amount: ");
        double amount = keyboard.nextDouble();
        account.withdraw(amount);
        System.out.printf("Withdrawal of $%.2f successfully made from account %d%n", amount,
accountNumber);
      } else {
        System.out.println("No account exist with AccountNumber: " + accountNumber);
      }
    }
```

```
   public static void applyInterest(Bank bank, Scanner keyboard) {
      // Get the account number
      System.out.print("\nPlease enter account number: ");
      int accountNumber = keyboard.nextInt();

      Account account = bank.getAccount(accountNumber);
      if (account! = null && account instanceof SavingsAccount) {
         ((SavingsAccount) account).applyInterest();
         System.out.printf("Interest applied to account %d%n", accountNumber);
      } else {
         System.out.println("No or invalid savings account exist with AccountNumber: " +
accountNumber);
 }
   }

   public static Account createAccount(Scanner keyboard) {
      Account account = null;
      int choice = accountMenu(keyboard);

      int accountNumber;
      System.out.print("Enter Account Number: ");
      accountNumber = keyboard.nextInt();

      if (choice == 1) { // checking account
         System.out.print("Enter Transaction Fee: ");
         double fee = keyboard.nextDouble();
```

```
            account = new CheckingAccount(accountNumber, fee);
            System.out.printf("Checking Account created with Account Number: %d and Transaction Fee: $%.2f%n", accountNumber, fee);
        } else { // savings account
            System.out.print("Please enter Interest Rate: ");
            double ir = keyboard.nextDouble();
            account = new SavingsAccount(accountNumber, ir);
            System.out.printf("Savings Account created with Account Number: %d and Interest Rate: %.2f%%%n", accountNumber, ir);
        }
        return account;
    }

    public static void printTransactionHistory(Bank bank, Scanner keyboard) {
        // Get the account number
        System.out.print("\nPlease enter account number: ");
        int accountNumber = keyboard.nextInt();

        Account account = bank.getAccount(accountNumber);
        if (account != null) {
            List<Transaction> transactions = account.getTransactions();
            System.out.println("Transaction History for Account " + accountNumber + ":");
            for (Transaction transaction : transactions) {
                System.out.println(transaction);
            }
        } else {
            System.out.println("No account exist with AccountNumber: " + accountNumber);
        }
```

```
    }
    /**
     * Menu to display options and get the user's selection
     */

    public static int menu (Scanner keyboard) {
        System.out.println("Bank Account Menu");
        System.out.println("1. Create New Account");
        System.out.println("2. Deposit Funds");
        System.out.println("3. Withdraw Funds");
        System.out.println("4. Apply Interest");
        System.out.println("5. Show Transaction History");
        System.out.println("6. Quit");

        int choice;

        do {
            System.out.print("Enter choice: ");
            choice = keyboard.nextInt();
        } while (choice < 1 || choice > 6);

        return choice;
    }
}
```

Bank Class

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Bank {
   private Account [] accounts;
   private List<Transaction> [] transactionHistory;
   private int numAccounts;

   @SuppressWarnings("unchecked")
public Bank (int capacity) {
   if (capacity <= 0) {
      throw new IllegalArgumentException("Capacity must be greater than 0");
   }
   this.accounts = new Account[capacity];
   this.transactionHistory = new ArrayList[capacity];
   this.numAccounts = 0;
}

   public void addAccount(Account account) {
      if (numAccounts < accounts.length) {
         accounts[numAccounts] = account;
         transactionHistory[numAccounts] = new ArrayList<> ();
         numAccounts++;
      } else {
```

```
            System.out.println("Maximum account limit reached. Unable to add new accounts.");
        }
    }
    public Account getAccount(int accountNumber) {
        for (int i = 0; i < numAccounts; i++) {
            if (accounts[i]. getAccountNumber() == accountNumber) {
                return accounts[i];
            }
        }
        return null; // Account not found
    }

    public Account [] getAllAccounts() {
        return Arrays.copyOf(accounts, numAccounts);
    }

    public void deposit (int accountNumber, double amount) {
        Account account = getAccount(accountNumber);
        if (account! = null) {
            account.deposit(amount);
            transactionHistory[getAccountIndex(accountNumber)]. add (new Transaction
(transactionHistory.length + 1, accountNumber, amount, TransactionType.DEPOSIT));
        } else {
            System.out.println("Account not found.");
        }
    }
```

```
    public void withdraw (int accountNumber, double amount) {
        Account account = getAccount(accountNumber);
        if (account! = null) {
            account.withdraw(amount);
            transactionHistory[getAccountIndex(accountNumber)]. add (new Transaction
(transactionHistory.length + 1, accountNumber, amount, TransactionType.WITHDRAWAL));
        } else {
            System.out.println("Account not found.");
        }
    }
    public List<Transaction> getTransactionHistory(int accountNumber) {
        int index = getAccountIndex(accountNumber);
        if (index! = -1) {
            return transactionHistory[index];
        }
        return null;
    }
    private int getAccountIndex(int accountNumber) {
        for (int i = 0; i < numAccounts; i++) {
            if (accounts[i]. getAccountNumber () == accountNumber) {
                return i;
            }
        }
        return -1; // Account not found
    }
    }
```

Account Class

```
import java.util.ArrayList;
import java.util.List;

public abstract class Account {
   protected int accountNumber;
   protected double balance;
   private List<Transaction> transactions;

   public Account (int accountNumber) {
      this.accountNumber = accountNumber;
      this.balance = 0;
      this.transactions = new ArrayList<>();
   }

    public Account () {
      this (000);
    }

   public double getBalance() {
      return balance;
   }

   public int getAccountNumber() {
      return accountNumber;
   }
```

```
    public void recordTransaction(Transaction transaction) {
        transactions.add(transaction);
    }

    public List<Transaction> getTransactions() {
        return new ArrayList<>(transactions);
    }

    public abstract void deposit (double amount);
    public abstract void withdraw (double amount);
}
```

CheckingAccount Class

```
public class CheckingAccount extends Account {

    // Default Transaction Fee
    private static double FEE = 100.568;
    private int transactionId;

    // default constructor
    public CheckingAccount() {
        super ();
    }
```

```
/**
 * Parameter constructor to initialize CheckingAccount
 * with a custom Account Number and a Customer Transaction
 * Fee.
 */
public CheckingAccount(int accountNumber, double fee) {
   super(accountNumber);
   FEE = fee;
}
/**
 * Function to deposit funds into the account as long as the amount
 *  parameter is > 0
 * Apply Transaction fee for the CheckingAccount
 */
@Override
public void deposit (double amount) {
   // First Check amount
   if (amount > 0) {
      balance += amount;
      System.out.printf("Amount %.2f deposited%n", amount);

      // Apply Transaction Fee
      balance -= FEE;
      System.out.printf("Fee %.2f Applied%n", FEE);
      System.out.printf("Current Balance is: %.2f%n", balance);

      // Record deposit transaction
```

```
        recordTransaction(new Transaction(transactionId++, getAccountNumber(), amount,
TransactionType.DEPOSIT));
      } else {
        System.out.println("A negative amount cannot be deposited");
      }
    }

    /**
     * Function to withdraw funds from the Account as long as 1. Amount to withdraw
     * must be > 0 2. Amount to withdraw must be <= balance
     */
    @Override
    public void withdraw (double amount) {
      // Same check
      if (amount > 0) {
        // Check sufficient balance
        if ((amount + FEE) <= balance) {
          System.out.printf("Amount of %.2f withdrawn from Account%n", amount);
          balance -= amount;
          balance -= FEE;
          System.out.printf("Fee of %.2f applied%n", FEE);
          System.out.printf("Current Balance is: %.2f%n", balance);

          // Record withdrawal transaction
        recordTransaction(new Transaction(transactionId++, getAccountNumber(), amount,
TransactionType.WITHDRAWAL));
        }
      } else {
```

```
            System.out.println("Negative amount cannot be withdrawn!");
        }
    }
}
```

SavingsAccount Class

```
public class SavingsAccount extends Account {

    // interest rate
    private double interestRate;
    private int transactionId;

    // default constructor
    public SavingsAccount() {
        super ();
    }

    /**
     * Parameter constructor to initialize Savings account with a customer account
     * number and interest rate
     */
    public SavingsAccount(int accountNumber, double interestRate) {
        super(accountNumber);
        this.interestRate = interestRate;
```

```
    }

    // getter function
    public double getInterestRate() {
        return this.interestRate;
    }

    public void setInterestRate(double interestRate) {
        this.interestRate = interestRate;
    }

    public double calcInterest() {
        return balance * interestRate;
    }

    public void applyInterest() {
        double interest = calcInterest();
        System.out.printf("Interest amount %.2f added to balance%n", interest);
        deposit(interest);

        // Record interest transaction
        recordTransaction(new Transaction(transactionId++, getAccountNumber(), interest,
TransactionType.INTEREST));
    }

    /**
     * Function to deposit funds into the account as long as the amount
```

```
     * parameter is > 0
     * Apply Transaction fee for the CheckingAccount
     */

    @Override
    public void deposit (double amount) {
        // First Check amount
        if (amount > 0) {
            balance += amount;
            System.out.printf("Amount %.2f deposited%n", amount);
            System.out.printf("Current Balance is: %.2f%n", balance);

            // Record deposit transaction
            recordTransaction(new Transaction(transactionId++, getAccountNumber(), amount,
TransactionType.DEPOSIT));
        } else {
            System.out.println("A negative amount cannot be deposited");
        }
    }

    /**
     * Function to withdraw funds from the Account as long as
     * 1. Amount to withdraw
     * must be > 0
     * 2. Amount to withdraw must be <= balance
     */
    @Override
```

```
    public void withdraw (double amount) {
        // Same check
        if (amount > 0) {
            // Check sufficient balance
            if (amount <= balance) {
                System.out.printf("Amount of %.2f withdrawn from Account%n", amount);
                balance -= amount;
                System.out.printf("Current Balance is: %.2f%n", balance);

                // Record withdrawal transaction
                recordTransaction(new Transaction(transactionId++, getAccountNumber(), amount,
TransactionType.WITHDRAWAL));
            }
        } else {
            System.out.println("Negative amount cannot be withdrawn!");
        }
    }
}
```

Transaction Class

```
import java.time.LocalDateTime;
import java.time.ZoneId;

public class Transaction {
    private int transactionId;
    private int accountNumber;
```

```
    private double amount;
    private TransactionType type;
    private LocalDateTime timestamp;

    public Transaction (int transactionId, int accountNumber, double amount, TransactionType type) {
        this.transactionId = transactionId;
        this.accountNumber = accountNumber;
        this.amount = amount;
        this.type = type;
        this.timestamp = LocalDateTime.now(ZoneId.of("Asia/Dhaka")); // Set to Bangladesh time zone
    }

    // Getters for transaction details
    public int getTransactionId() {
        return transactionId;
    }

    public int getAccountNumber() {
        return accountNumber;
    }

    public double getAmount() {
        return amount;
    }

    public TransactionType getType() {
```

```
        return type;
    }

    public LocalDateTime getTimestamp() {
        return timestamp;
    }

    @Override
    public String toString() {
        return "Transaction ID: " + transactionId +
                ", Type: " + type +
                ", Amount: $" + amount +
                ", Timestamp: " + timestamp;
    }
}
```

TransactionType Enum

```
public enum TransactionType {
    DEPOSIT,
    WITHDRAWAL, INTEREST
}
```

Input:

Bank Account Menu

1. Create New Account

2. Deposit Funds

3. Withdraw Funds

4. Apply Interest

5. Show Transaction History

6. Quit

Enter choice: 1

Select Account Type

1. Checking Account

2. Savings Account

Enter choice: 1

Enter Account Number: 771439

Enter Transaction Fee: 3000

Bank Account Menu

1. Create New Account

2. Deposit Funds

3. Withdraw Funds

4. Apply Interest

5. Show Transaction History

6. Quit

Enter choice: 2

Please enter account number: 771439

Please enter Deposit Amount: 70000

Bank Account Menu

1. Create New Account

2. Deposit Funds

3. Withdraw Funds

4. Apply Interest

5. Show Transaction History

6. Quit

Enter choice: 5

Please enter account number: 771439

Output:

```
Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Show Transaction History
6. Quit
Enter choice: 1

Select Account Type
1. Checking Account
2. Savings Account
Enter choice: 1
Enter Account Number: 771439
Enter Transaction Fee: 3000
Checking Account created with Account Number: 771439 and Transaction Fee: $3000.00

Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Show Transaction History
6. Quit
Enter choice: 2


Please enter account number: 771439
Please enter Deposit Amount: 70000
Amount 70000.00 deposited
Fee 3000.00 Applied
Current Balance is: 67000.00
Deposit of $70000.00 successfully made to account 771439

Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Show Transaction History
6. Quit
Enter choice: 5


Please enter account number: 771439
Transaction History for Account 771439:
Transaction ID: 0, Type: DEPOSIT, Amount: $70000.0, Timestamp: 2024-03-10T01:15:47.965374800
```

This scenario tests the functionality of depositing funds into a checking account and applying transaction fees in the Java-based bank management application we developed.

Discussion:

In this report, we implemented a Java application based on Object-Oriented Programming (OOP) concepts, specifically focusing on developing a Bank Management System. The system is designed to efficiently manage accounts and transactions within a banking environment. Leveraging OOP principles such as encapsulation, inheritance, polymorphism, and abstraction, our application embodies modularity, scalability, and maintainability. Through the implementation of classes like Bank, Account (Abstract), CheckingAccount, SavingsAccount, Transaction, TransactionType Enum, and AccountDriver, we ensure a clear separation of concerns and facilitate easy expansion and maintenance of the system. The system's user-friendly interface, implemented in the AccountDriver class, enhances the overall user experience by providing intuitive menus and input handling for banking operations.

However, challenges arose in designing the class structure and managing class relationships to maintain data integrity.Despite facing challenges we successfully developed a well-structured and functional Bank Management System that showcases the practical application of Object-Oriented Programming (OOP) principles in business solutions.

Through the implementation of this Java application, we gained valuable insights into the practical application of Object-Oriented Programming (OOP) principles.By overcoming challenges and achieving our goals, we deepened our understanding of OOP concepts and their relevance in software development. This experience reinforced the importance of careful design and problem-solving skills in building effective and scalable systems.